adafruit learning system

Mini Thermal Receipt Printer

Created by Phillip Burgess

Last updated on 2015-08-04 03:20:13 PM EDT

Guide Contents

Overview

Add a mini printer to any microcontroller project with this very cute thermal printer. Also known as receipt printers, they're what you see at the ATM or grocery store. Now you can embed a little printer of your own into an enclosure. This printer is ideal for interfacing with a microcontroller, you simply need a 3.3V to 5V TTL serial output from your microcontroller to print text, barcodes, bitmap graphics, even a QR code!

This printer uses very common 2.25" wide thermal paper, available in the Adafruit shop (http://adafru.it/599) or any office or stationery supply store. Up to 50 feet of paper can fit in the bay. You will also need a 5 Volt to 9 Volt regulated DC power supply that can provide 1.5 Amps or more during high-current printing — our 5V 2A power supply will work very nicely (http://adafru.it/276).

You can pick up a thermal printer pack including printer, paper, power supply and terminal-block adapter in the Adafruit shop! (http://adafru.it/600)

Of course, we wouldn't leave you with a datasheet and a "good luck!" — this tutorial and matching

Arduino library demonstrate the following:

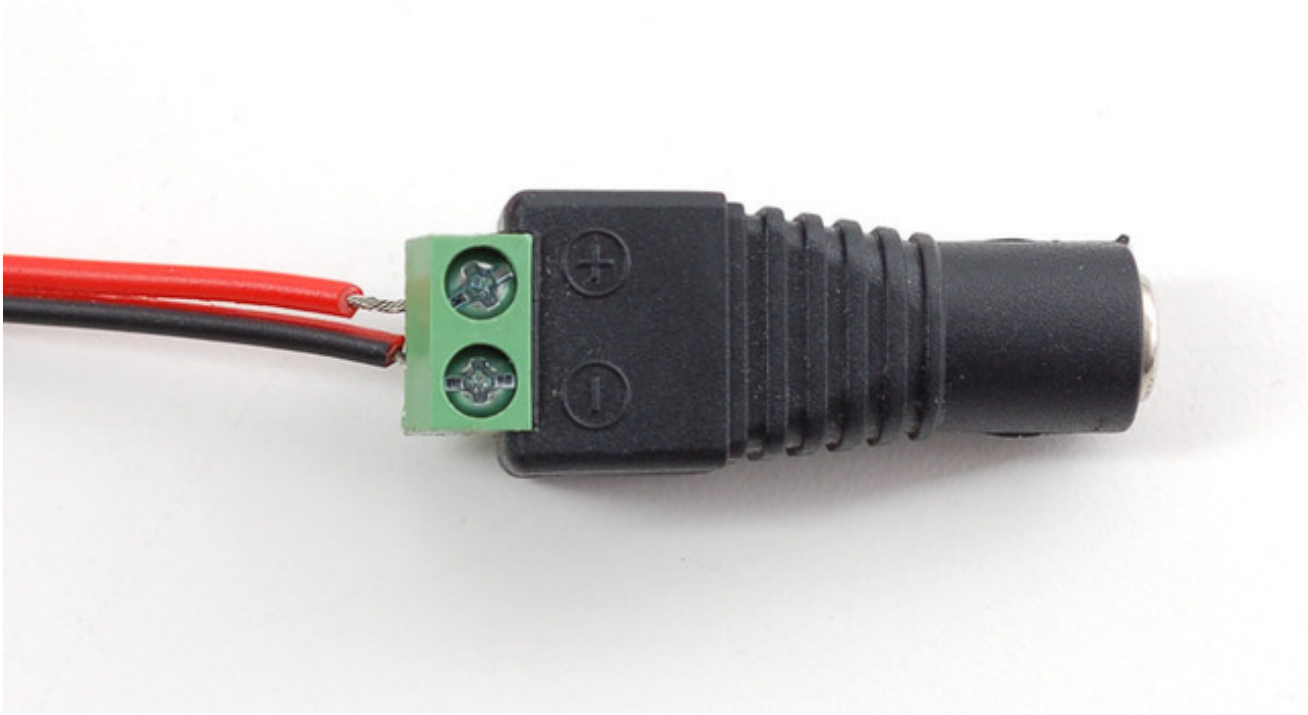
- Printing with small, medium and large text
- **Bold,** underline and inverted text
- Variable line spacing
- Left, center and right justification
- Barcodes in the following standard formats: **UPC A, UPC E, EAN13, EAN8, CODE39, I25, CODEBAR, CODE93, CODE128, CODE11** and **MSI** - with adjustable barcode height
- Custom monochrome bitmap graphics
- How to print a QR code

Power

These printers use a thermal head to heat the special receipt paper and draw images and text. That makes the printer very small — there's no moving ink head — but it means they require a lot of power. This printer in particular requires **5 to 9 Volts, 1.5 Amps current!** That means you will need a fairly beefy supply and you **cannot** run it off of USB power. An external adapter is *required!*

We suggest using the 5V 2A power supply in our shop (http://adafru.it/276). It's got plenty of power to keep the printer happy and you can also use it to run some microcontrollers or sensors off of the remaining 500 mA current that is not required by the printer.

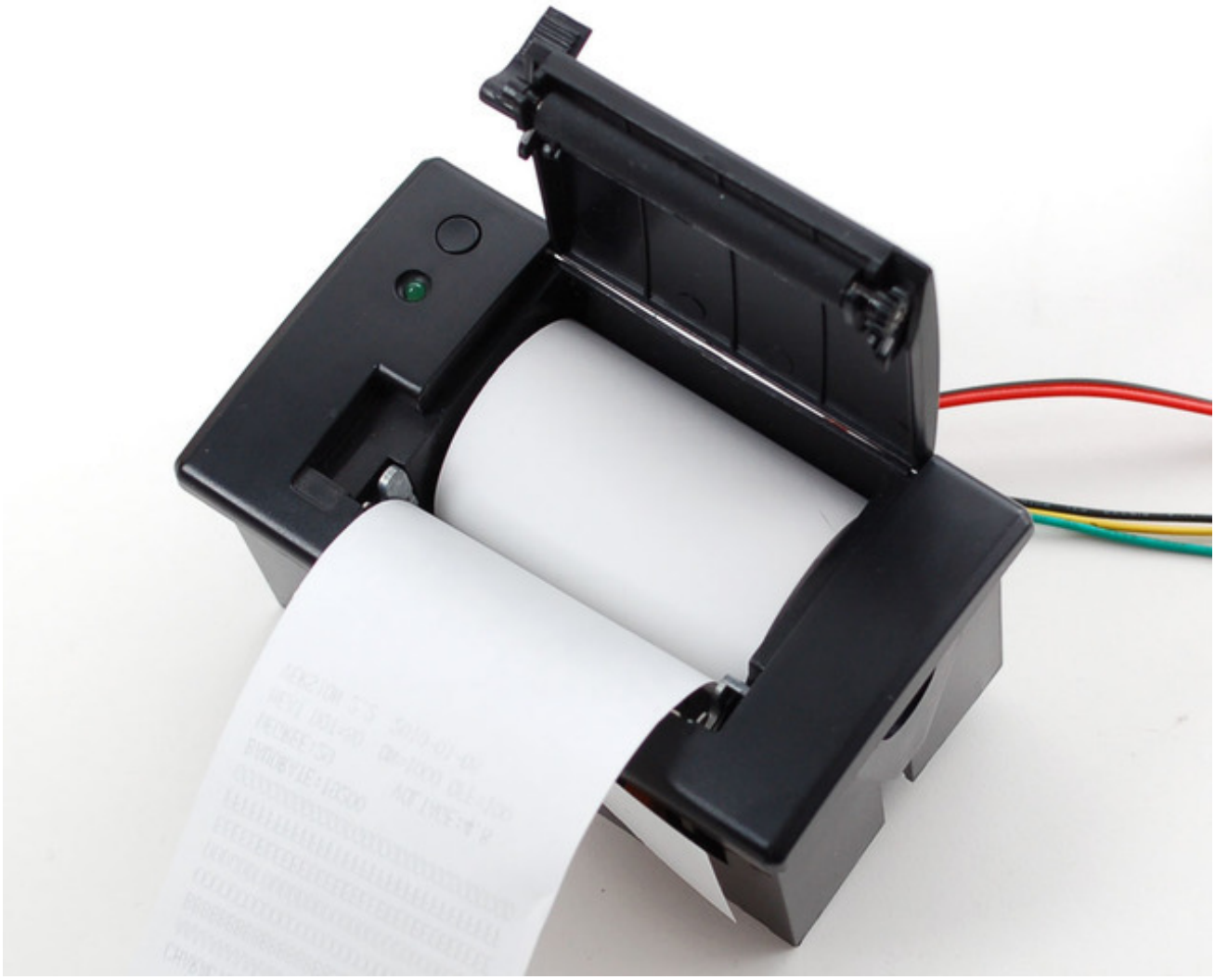
A quick way to power the printer is by just using a 2.1mm jack adapter (http://adafru.it/368), which you can attach to the printer's red/black wires:



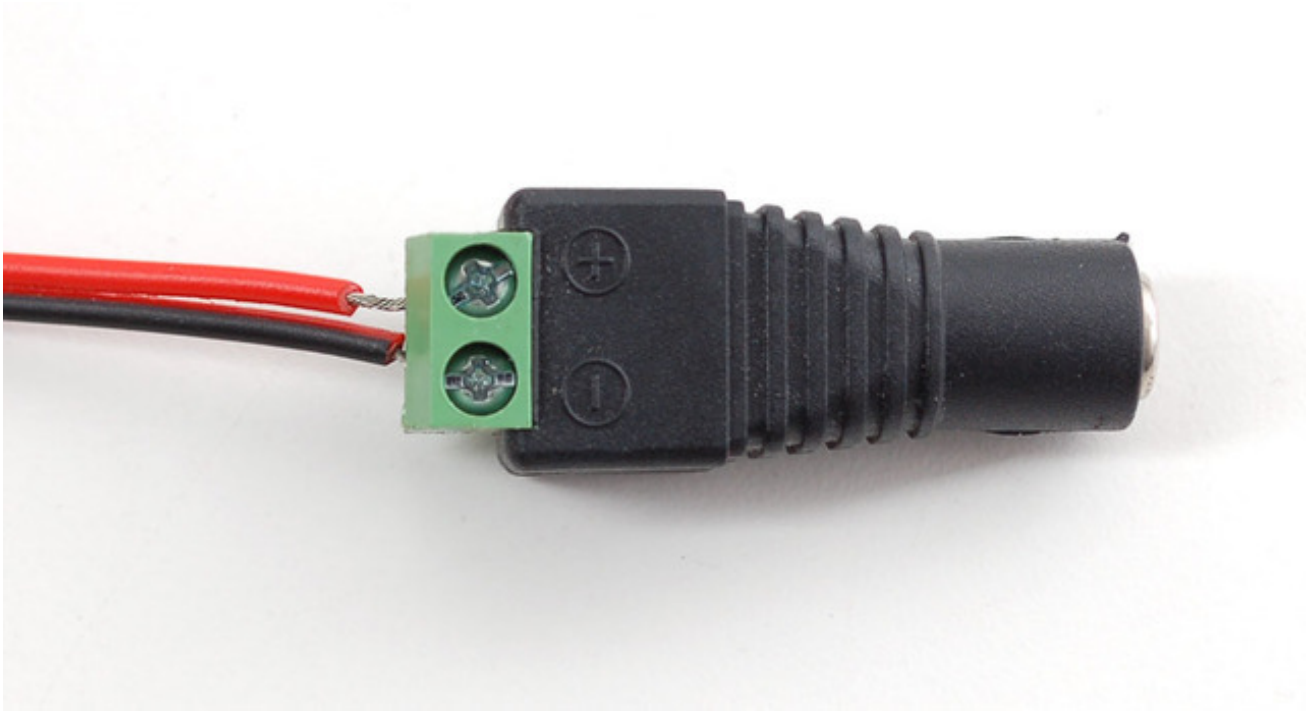
First Test

The first test you should do is to just make sure that the printer is running and you have the power wired up right.

First up, pull the little plastic tab up top to open up the paper holder. Then insert a roll of 57.5mm (2 1/4 inches) thermal paper into the bay as shown below. The optimal length of the paper will be 50 feet (about 15 meters) so try to pick up that size. Sometimes if you buy paper from an office supply shop, its a little longer, around 80 or 85 feet in which case you'll need to remove paper from the roll until its 1.5"/40mm in diameter and fits easily. Make sure that the paper doesn't bind or stick in the bay, it should rotate freely.



As previously described, power the printer using a 5V to 9V 1.5A or higher power supply, such as wiring up a 2.1mm DC power jack:

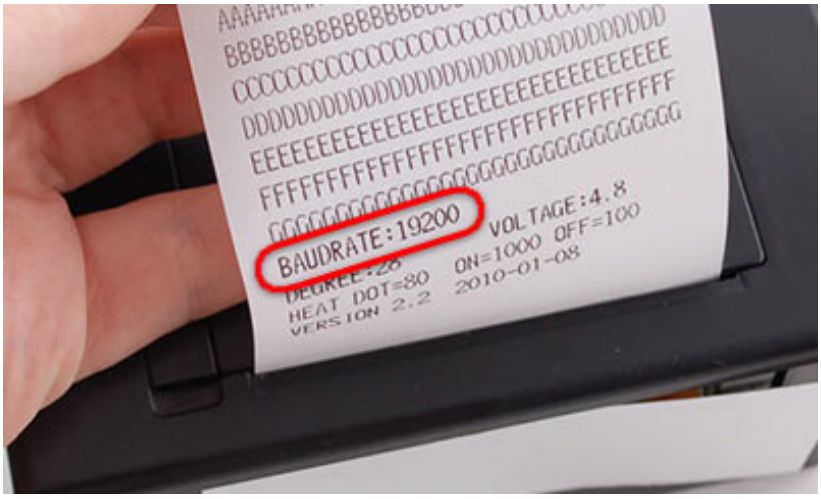


Hold down the button on the top of the printer while plugging in the power. You should see a receipt print out showing the font table and some diagnostics.

The Green LED should be blinking when the printer is powered. It will not be on steady!

If you don't get a printout, check that the paper is inserted correctly and not binding, that the power is correctly wired, power supply is plugged in, etc. Then try again, holding down the top button when connecting power.

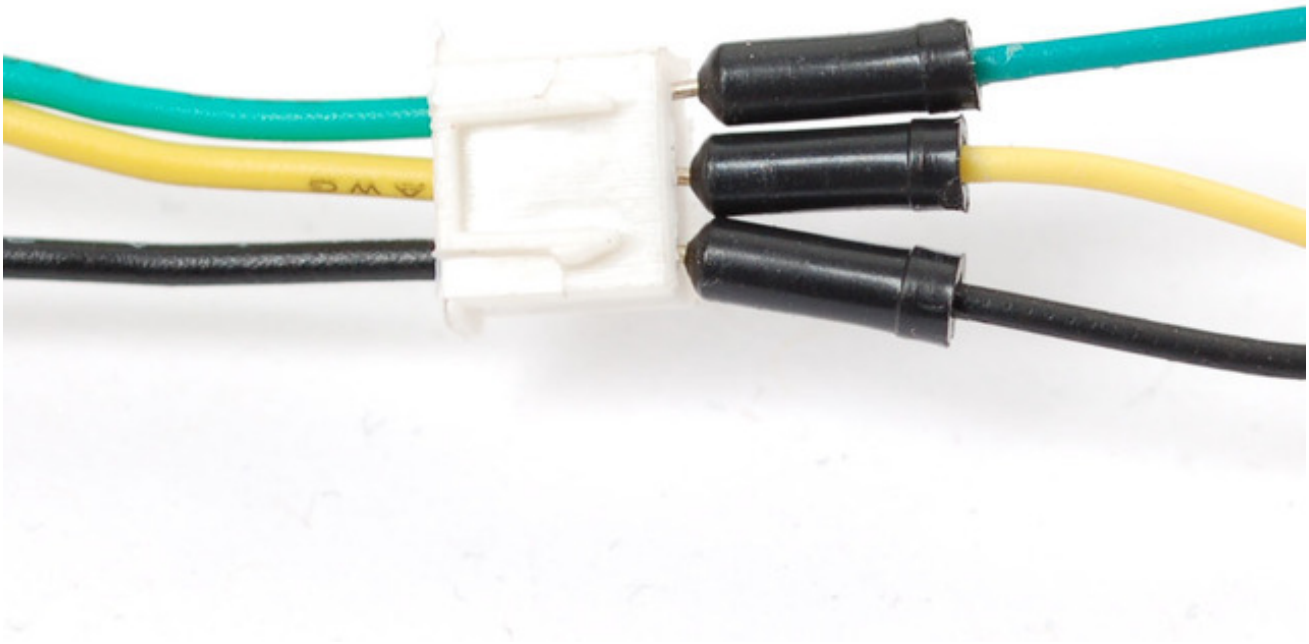
Note the baud rate on the test page. This may be 19200 or 9600. We'll need this number later:



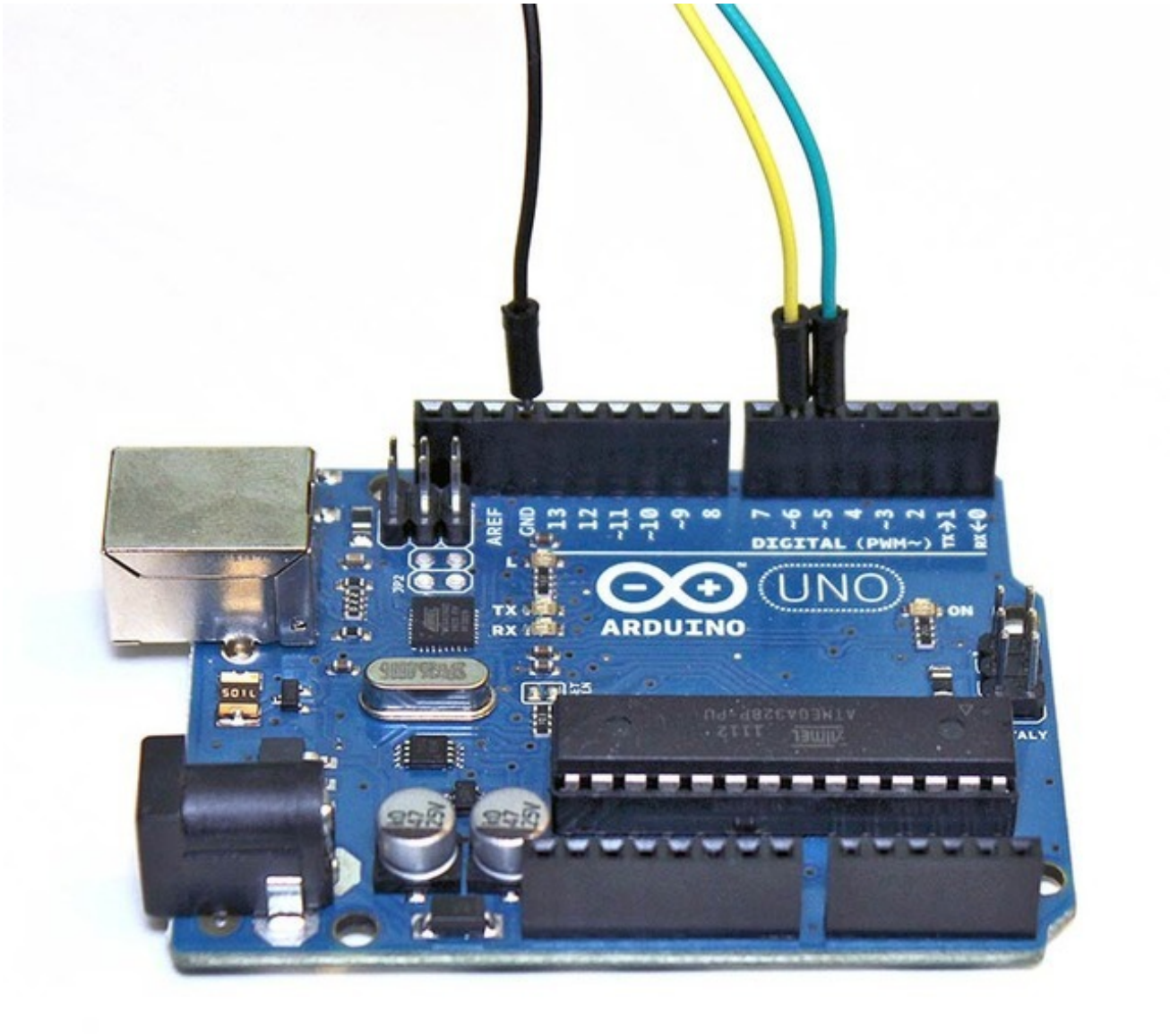
Microcontroller

To send data to the printer, we will use a 5V TTL serial connection. This is not the same as the 10V RS232 serial from a computer's 9-pin serial port — don't connect the printer directly to a standard PC port or you may damage it. It's possible to use something like an FTDI cable to talk to the printer, but we're going to assume that nearly everyone will want to use it with a microcontroller. This tutorial shows how to wire it up to an Arduino, and our example code is Arduino-compatible. Any microcontroller that can output TTL serial will work, with suitable adaptation to the code.

To start, we'll connect to the data cable of the printer, which has three wires: black, yellow and green. An easy way to connect is to simply press 22AWG or so wires of matching colors into the plug, then use those to extend the connection to an Arduino.



At the Arduino end, the **green** wire connects to **digital pin 5**, **yellow** goes to **digital pin 6** and **black** to any of the **GND** pins. You can change the digital pins later, but to match the example code, stick to this for now!



Now its time to download the Arduino library code. Visit the Adafruit Thermal Printer Library (http://adafru.it/aHt) on GitHub. To download, click the “ZIP” button near the top left, uncompress the ZIP file and rename the resulting uncompressed folder to **Adafruit_Thermal**. Confirm that this folder contains the files Adafruit_Thermal.cpp and Adafruit_Thermal.h, along with examples and other items. Place the Adafruit_Thermal library inside your Arduino libraries folder. We have a tutorial on library installation (http://adafru.it/aYG) to guide you through this.

If running an older, pre-1.0 version of the Arduino software, you’ll also need to install the NewSoftSerial library. Download it by clicking this link (http://adafru.it/aM7) and install it as you did the Thermal library. This is not necessary if running the latest Arduino software.

After installing the libraries, restart the Arduino IDE. You should now be able to access the sample code by navigating through menus in this order:
File→Sketchbook→Libraries→Adafruit_Thermal→printertest

If your printer test page shows 'BAUDRATE: 9600', you'll need to make a small change to the library source code. Using a text editor (Notepad, etc.) open the file Adafruit_Thermal.cpp and change this line:

```
#define BAUDRATE  19200
```

to this:

#define BAUDRATE 9600

Most printers arrive from the factory set for 19200 baud, but a few may be set to 9600. This will *not* negatively impact the performance of your unit! The speed of the paper through the printer is already much less than this and you will not see any difference…it's strictly a data protocol issue of getting the microcontroller and printer communicating.

OK, now you're finally ready to run the printer demo. Open up the Arduino IDE and select File→Sketchbook→Libraries→Adafruit_Thermal→printertest and upload the sketch to the Arduino. You should see the printer print out the example receipt which includes all the capabilities of the library.

If this does not work, first check that the printer and Arduino are both powered, and that the yellow, green and black wires are properly connected to the Arduino.

Printing Text

The thermal printer has a few handy things it can do, most of which are in the **A_printertest** sketch. These are shown in the image above. In order, starting from the top:

- Inverted text: this is invoked by calling **inverseOn()** — you will get text that's white-on-black instead of black-on-white. **inverseOff()** turns this off.
- Double height: this makes text thats extra tall, call **doubleHeightOn()** — likewise, turn off with **doubleHeightOff()**
- Left/Center/Right justified: this aligns text to the left or right edge of the page, or centered. You can set the alignment by calling **justify('R')** (for right-justified), **justify('C')** (for centered) or **justify('L')** (for left-justified). Left-justified is the default state.
- **Bold** text: makes it stand out a bit more, enable with **boldOn()** and turn off with **boldOff()**
- Underlined text: makes it stand out a bit more, enable with **underlineOn()** and turn off with **underlineOff()**
- Large/Medium/Small text: by default we use small, medium is twice as tall, large is twice as wide/tall. Set the size with **setSize('L')**, **setSize('M')** or **setSize('S')**
- Line spacing: you can change the space *between* lines of text by

calling **setLineHeight()** where **numpix** is the number of pixels. The minimum is 24 (no extra space between lines), the default spacing is 32, and double-spaced text would be 64.

Look through the source of the A_printertest sketch to see these used in context.

Bitmap Printing

This printer can print out bitmaps, which can add a touch of class to a receipt with your logo or similar.

The first step is to get the logo prepared. The printer can only do monochrome (1-bit) images, and the maximum width is 384 pixels. We suggest starting with a small bitmap (100 pixels or less on each side) and then experimenting to get the size and look you want.

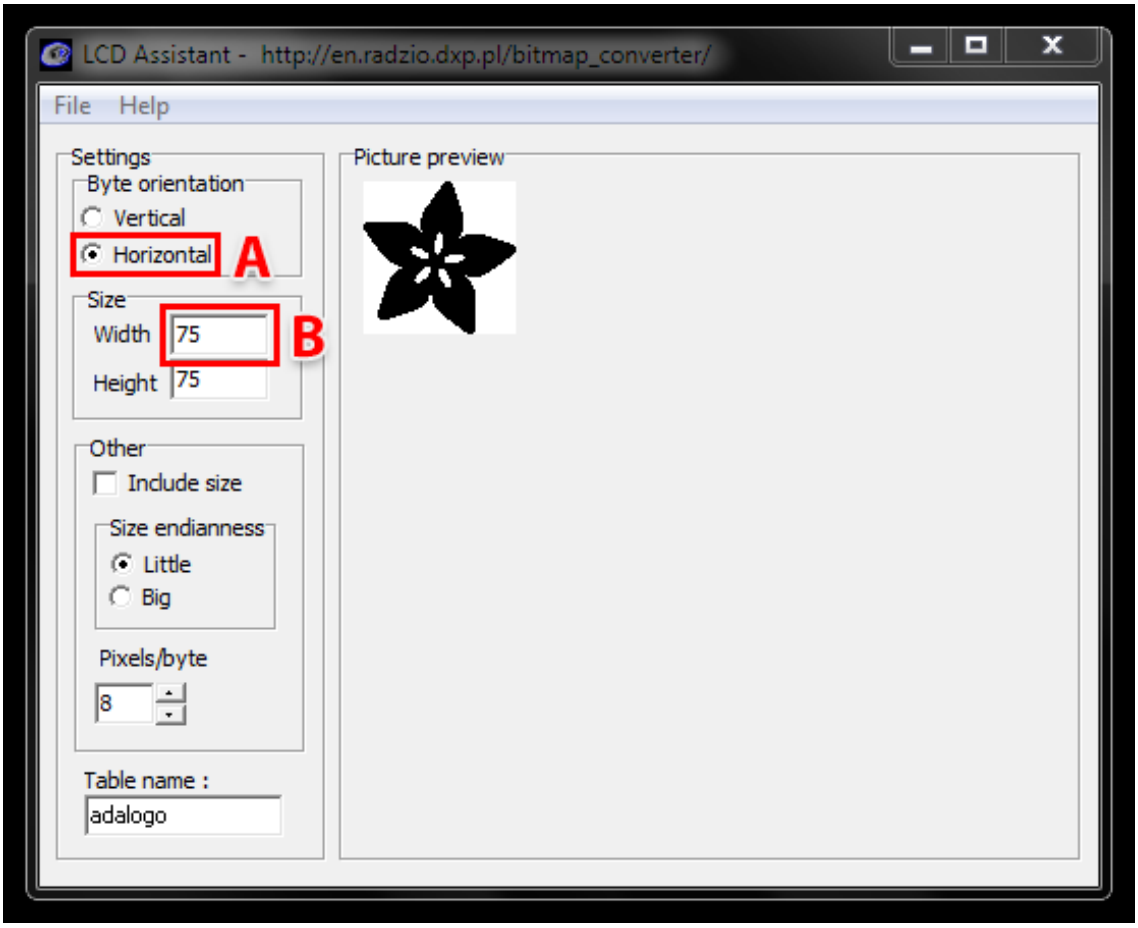
A few steps are required to prepare an image for printing. For Windows users, there's a nice graphical user interface for this. For Mac and Linux, different tools are used…not as visually slick, but they do the job well.

> These barcode printers shouldn't be confused with a LaserJet - they're not good at printing heavy/dense images with lots of black or they might stick and stall!

Windows

Use an image editing program to save your image as a 1-bit BMP — in Windows, the built-in **Paint** program will suffice.

Download, install and run **LCD Assistant (http://adafru.it/aPs)**. This program is for Windows only but does a really fantastic job! Load the BMP file you previously generated (in Paint, etc.). The file must be in BMP format — the software won't read PNG, GIF, etc. Then a couple of settings need to be adjusted…



First, in the “Byte orientation” section of the settings, select “Horizontal” (item A in the image above).

Second (item B above), you may need to change the Width setting. Because this software (and the thermal printer) handle images in horizontal groups of eight pixels, if the image width is not a multiple of 8, it will be truncated (cropped) to the nearest smaller 8-pixel boundary. For example, with the 75 pixel wide image above, the output will be cropped to only 72 pixels wide, losing some data from the right edge. To avoid this, **increase this number to the next multiple of 8** (that would be 80 for the example above), and the output will be padded with blank pixels to cover the gap. Remember the number you use here, you’ll need it later.

The image height does not need to be adjusted this way, only width.

Set the table name to something short but descriptive (e.g. “adalogo” above), then select Save Output from the File menu. Give the file a similarly brief but descriptive name, ending in “.h” (e.g. “adalogo.h”).

To get this file into your Arduino sketch, select “Add File…” from the Sketch menu. This will add a new tab to your code. Your original code is still there under the leftmost tab.

A couple of small changes are now needed in both tabs. First, at the top of the file containing the

new table data, change “const unsigned char” to “static const uint8_t PROGMEM” as shown below:

Next, in the tab containing the main body of your code, add an “include” statement to reference the new file:

```
#include "adalogo.h"
```

Check the **A_printertest** example sketch if you’re not sure how to include the code properly.

You can now output the image by calling **printBitmap(width, height, tablename)**, where **width** and **height** are the dimensions of the image in pixels (if you changed the image width to a multiple of 8 as previously described, use that number, not the original image size), and **tablename** is the name of the array in the new tab (e.g. “adalogo” above).

Having a graphical user interface is nice, but some of these extra steps can be confusing and error-prone. If you prefer, the technique below for Mac and Linux works in Windows as well.

Mac and Linux

The conversion tool for Mac and Linux doesn't include a fancy GUI, but it works well and avoids several steps (and potential mis-steps). The source image doesn't need to be in BMP format — most image formats can be read natively — and the output can be added to a sketch with no further editing. It works for Windows as well, if you'd rather use this method.

First, if you don't already have the **Processing** language installed, download it from processing.org (http://adafru.it/aPt). Processing looks almost exactly like the Arduino IDE, but it's for writing code for your normal computer, not a microcontroller. This can be a little confusing to first-timers, so if something doesn't seem to compile, make sure you're running code in the right environment: *Arduino* for for the Arduino board, *Processing* for your computer.

This code runs in Processing 2.x, the latest version of Processing available from their website. Don't use the older Processing 1.5.x version!

The Adafruit_Thermal library folder that you previously downloaded contains a sub-folder

called **processing**. Inside that is a sketch called **bitmapImageConvert.pde**. Load this into Processing and press RUN (the triangle button).

You'll be prompted to select an image using the system's standard file selection dialog. The program runs for just a brief instant, and will create a new file alongside the original image file. For example, if you selected an image called "adalogo.png", there will be a new file called "adalogo.h" in the same location. This file contains code to add to your Arduino sketch. You shouldn't need to edit this file unless you want to change the variable names within.

To get this file into your Arduino sketch, select "Add File…" from the Sketch menu. This will add a new tab to your code. Your original code is still there under the leftmost tab.

Next, in the tab containing the main body of your code, add an "include" statement to reference the new file:

```
#include "adalogo.h"
```

Check the **A_printertest** example sketch if you're not sure how to include the code properly.

If the source image was called adalogo.png, then the resulting .h file (adalogo.h) will contain three values called adalogo_width, adalogo_height and adalogo_data, which can be passed directly and in-order to the printBitmap() function, like this:

```
printBitmap(adalogo_width, adalogo_height, adalogo_data);
```

Barcode Printing

Thermal printers are really good at printing barcodes! This printer supports 11 different codes - **UPC A, UPC E, EAN13, EAN8, CODE39, I25, CODEBAR, CODE93, CODE128, CODE11** and **MSI**. It only supports linear (1-D) barcodes, and can't generate 2-D barcodes like QR codes (although there is a hack you can do, see below!) Barcodes are generated "on the fly," which is nice — you can customize the height and data included quite easily.

You can make a barcode by calling **printBarcode("barcodedata", BARCODETYPE)**, where the first string is the data to encode (e.g. a UPC code) and **BARCODETYPE** can be **UPC_A, UPC_E, EAN13, EAN8, CODE39, I25, CODEBAR, CODE93, CODE128, CODE11** or **MSI**.

Some barcodes are very restricted — you can only put in 12 numbers, no characters. Others are very flexible and take nearly any character input. Please check out the wikipedia list detailing kinds of barcodes (http://adafru.it/aPq)to pick the right one for your application.

The available range of barcodes varies with the printer firmware revision. Check Adafruit_Thermal.h for a list of codes.

It's also possible to print QR codes, if you're willing to pre-generate them. This might be handy if you want to, let's say, include a URL on the receipt and the URL doesn't change. You can generate QR codes at many sites including this one. (http://adafru.it/aPr) Use the smallest QR code size. The image will be in PNG format, so if you're using the Windows LCD Assistant tool you'll need to convert it to BMP first (Windows Paint works for this). Then you can convert and embed this in your Arduino sketch as previously described.

Downloads

- Adafruit_Thermal library for Arduino (http://adafru.it/aHt).
- NewSoftSerial library (http://adafru.it/aM7) — needed ONLY if you're using Arduino 0023 or earlier (not 1.0 or later).
- LCD Assistant (http://adafru.it/aPs) — optional bitmap conversion utility for Windows.
- Processing (http://adafru.it/aMI) language — needed for bitmap conversion for Mac or Linux (and optionally Windows). DOWNLOAD VERSION 1.5.1, not the 2.0 beta.
- Thermal Printer User Manual (http://adafru.it/f0G).
- An older version of the Thermal Printer User Manual (http://adafru.it/aPu).
- Thermal Printer Product Sheet (http://adafru.it/aPv).

Troubleshooting!

I'm not getting it to work!

First thing to try is the power up test. You do not need an arduino for this

Make sure the paper is in the bay correctly, it should feed under and up so that the paper comes through the slot without bending

Hold down the button on the top, then plug in the printer to power. It should print out a test page

The green LED will not be on solid, it will blink once in a while, that's normal.

I'm having difficulty getting a printout

Make sure the paper is in the bay correctly, it should feed under and up so that the paper comes through the slot without bending

Make sure the paper roll is not 'stuck' feeling in the bay, it should rotate easily!

My Arduino sketch used to work, but doesn't compile now!

Some changes have been made to support a broader range of Arduino-like boards. Older code will require updating. Fortunately it's just a few lines around the global declarations and the setup() function.

Old syntax:

Adafruit_Thermal printer(RX_PIN, TX_PIN);

New syntax: declare a SoftwareSerial object and pass its address to the Adafruit_Thermal constructor, like so:

SoftwareSerial mySerial(RX_PIN, TX_PIN);

Adafruit_Thermal printer(&mySerial);

Then, in the setup() function:

mySerial.begin(19200);

printer.begin();

My sketch compiles, but the output is different than before.

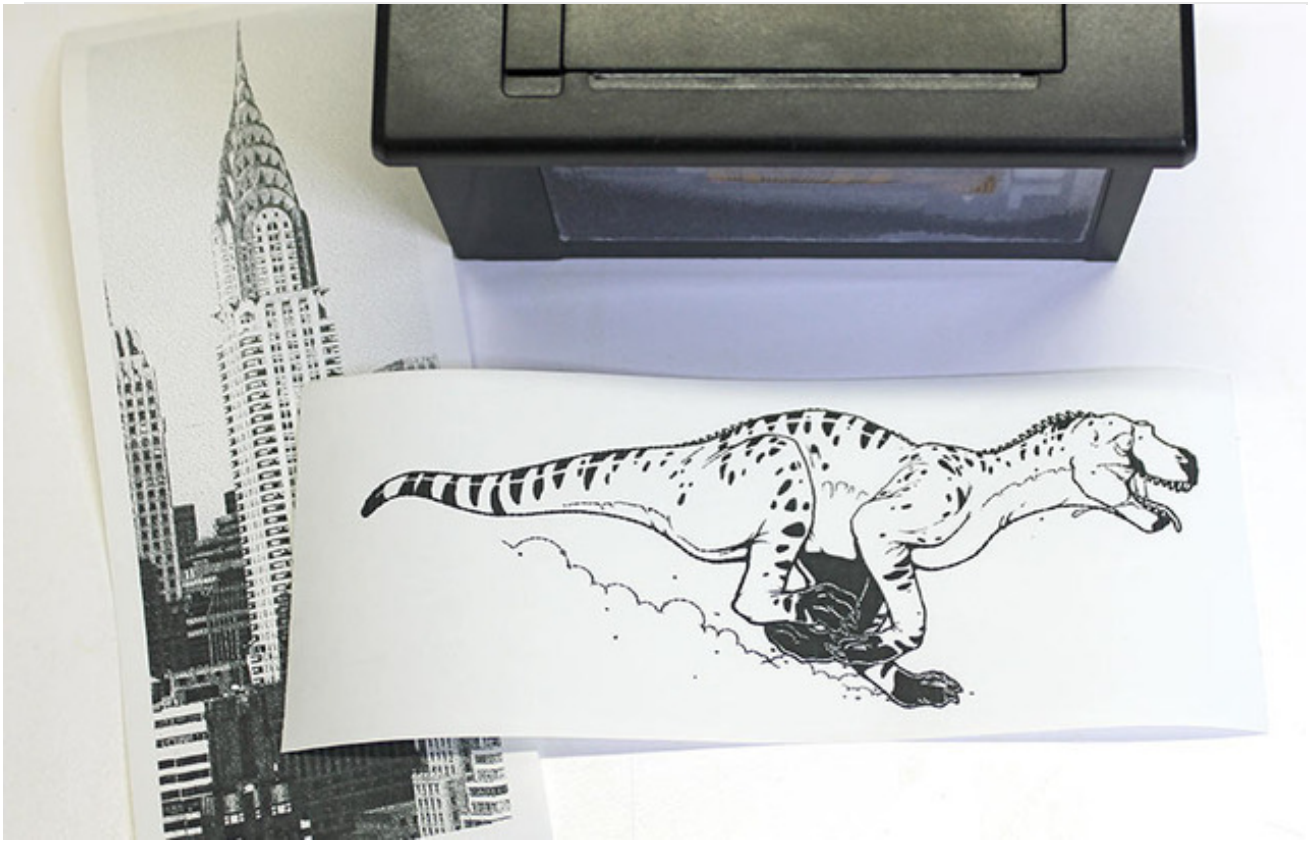
The printer's features and behavior have changed over various firmware releases.

First, check the PRINTER_FIRMWARE value in Adafruit_Thermal.h, make sure it matches the

value at the bottom of the printer test page (hold down the paper feed button when connecting power to print out a test page).

Some features just behave a little differently among releases…barcodes, line feeds, etc.…if you were previously using an older library or an older printer, you may need to tweak the code to produce the desired formatting with a current setup.

Hacking!



Look at those huge, razor-sharp image prints! You want some?

The following…

- Is an **undocumented** printer feature and is **NOT guaranteed** to work.
- Requires **modifying** your printer — a **warranty-voiding** operation! Continue at your own risk.

You should *only* attempt this if ***all*** of the following apply:

- Have first confirmed that the **printer works as expected** when operated through **conventional procedures**.
- Have a **genuine performance bottleneck** that cannot be adequately resolved by **adjusting the printer** timing and thermal settings first.
- Are comfortable **opening things** and **soldering**.

A modified printer is not eligible for a refund or exchange.

These printers have a limited serial receive buffer. Push bits to the printer faster than it can physically heat dots and feed paper, and you experience an “overflow” — bitmap images become garbled, text and formatting commands may be skipped.

The thermal printer library tries to throttle data to the printer at just the right rate. Too fast and an overflow occurs. Too slow and it wastes your time; the printer isn’t operating at peak throughput. This is an imperfect process…though we use very conservative timing estimates, the actual speed through the printer is impossible to predict…sometimes overflows *still* occur.

Hardware handshaking is a means by which a printer or other device can report to the microcontroller that it’s ready to receive more data, virtually eliminating buffer overflows while operating at peak throughput…the paper feed stops only when it physically absolutely must. Optimal performance.

It appears that some varieties of these thermal printers support hardware handshaking (e.g. firmware v2.64, 2.68). This is barely mentioned in the datasheet, and in fact **there isn’t even a physical connection for this on the outside of the printer.** A little surgery is in order…

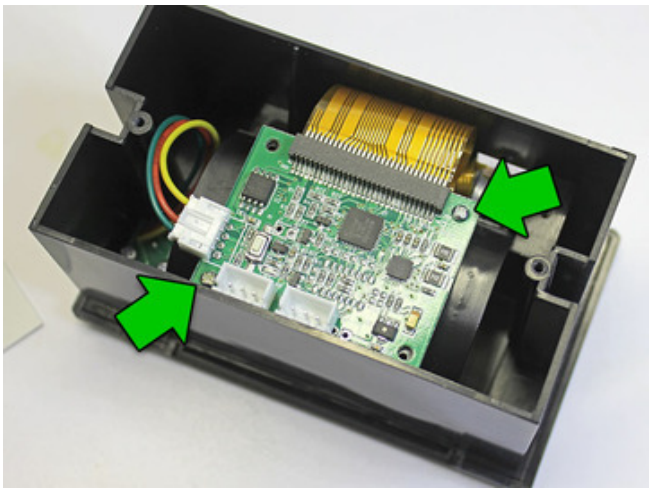
Parts and Tools Needed

- Small Phillips head screwdriver
- Pliers
- Soldering iron and related paraphernalia
- A bit of wire…but ideally a female jumper wire

Procedure

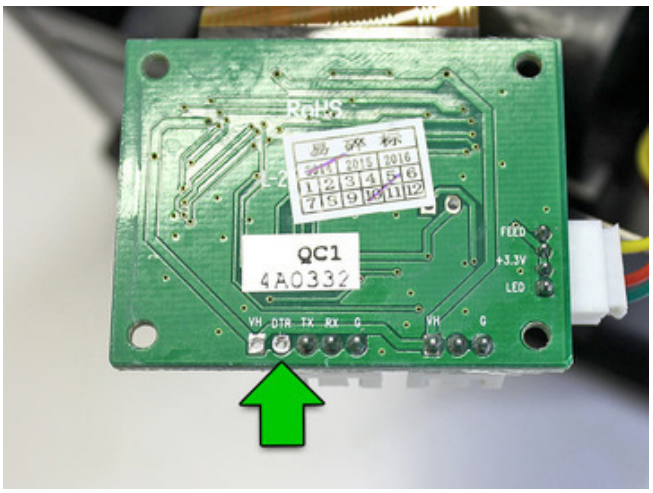
Unplug all cables, turn the printer over and remove the two small Phillips screws.

Take the back plate off, then remove the two (or



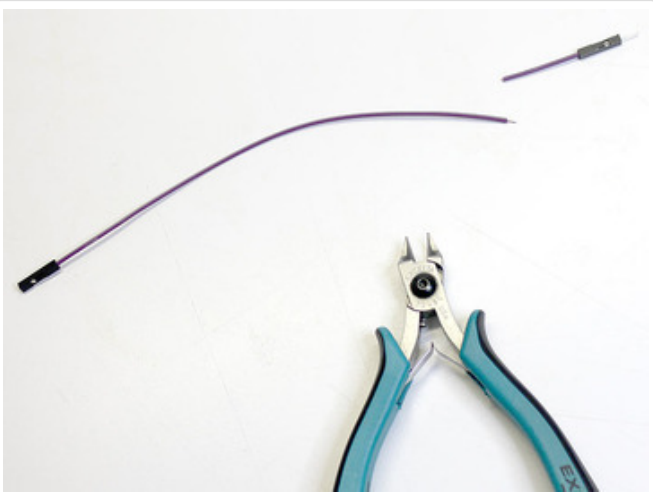
sometimes four) Phillips screws holding the circuit board in place.

These screws are a little smaller than the back-holding ones…don't get them mixed up!



Carefully, so as not to unseat or unplug the connectors, turn the circuit board over and look for the unpopulated via labeled "DTR."

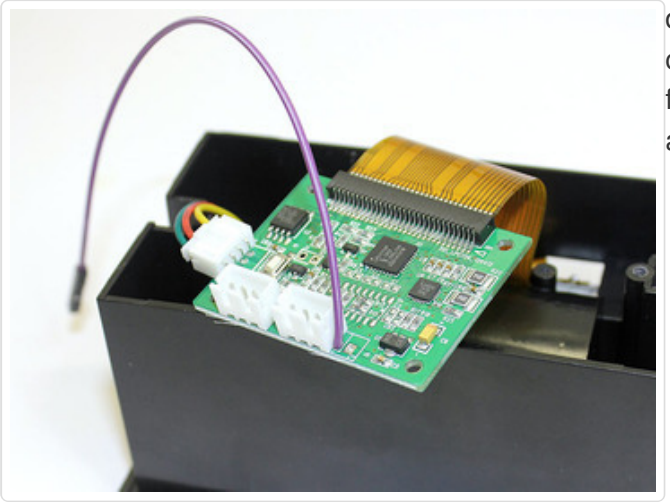
There are some other interesting solder points in here, if you're so inclined. "HV" is the raw 5–9 Volts from the power supply. On the right is a 3.3V pin, though I don't know the available current. Conceivably one could bring these out to reduce overall cabling in a project…or even install a tiny microcontroller right inside!



Cut an end off a female jumper wire and strip & tin the end.

This will be hanging out of the printer…so a *female* jumper prevents accidental contact with things if you're not using the connection. If you only have regular wire, that's fine, just be careful not to leave a bare end dangling.

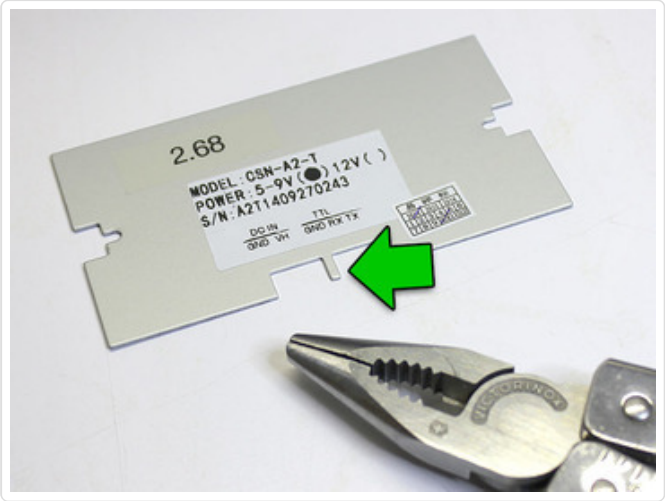
Solder the wire to the DTR pad. Top, bottom,

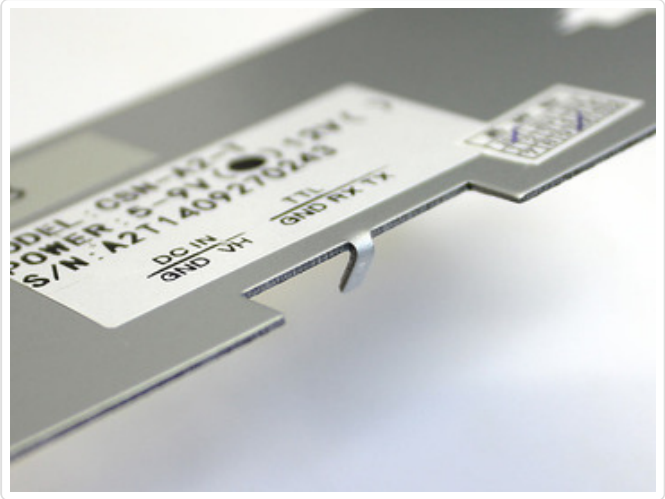


doesn't matter…it's right up against the serial connection plug, so use whatever path works best for you, there's ample room for routing the wire around either way.

Pedants may note that this isn't technically a DTR pin, but rather CTS. It's long-standing thing among printer manufacturers…apparently the misnomer was made decades ago but has stuck for consistency.

On the back plate, there's a small metal "finger" between the serial and power sockets. Using pliers, this can be bent back to provide an exit route for the DTR wire.





Screw the controller board back in place (check that neither of the cables has come unseated), routing the DTR wire around between the two sockets, then screw the back on.

Finished with the hardware!

You can then reconnect the power and serial sockets, and wire those up as before.

Use a jumper wire to connect DTR to any available Arduino pin. In our examples, we'll use digital pin 4.

The printer electronics operate at 3.3V, so no level shifting is needed with 3.3V boards (Arduino Due, etc.)…this can safely be connected directly.

Code Changes

Just one line…the Adafruit_Thermal constructor…needs changing. It can accept an optional parameter, a pin number to use for DTR:

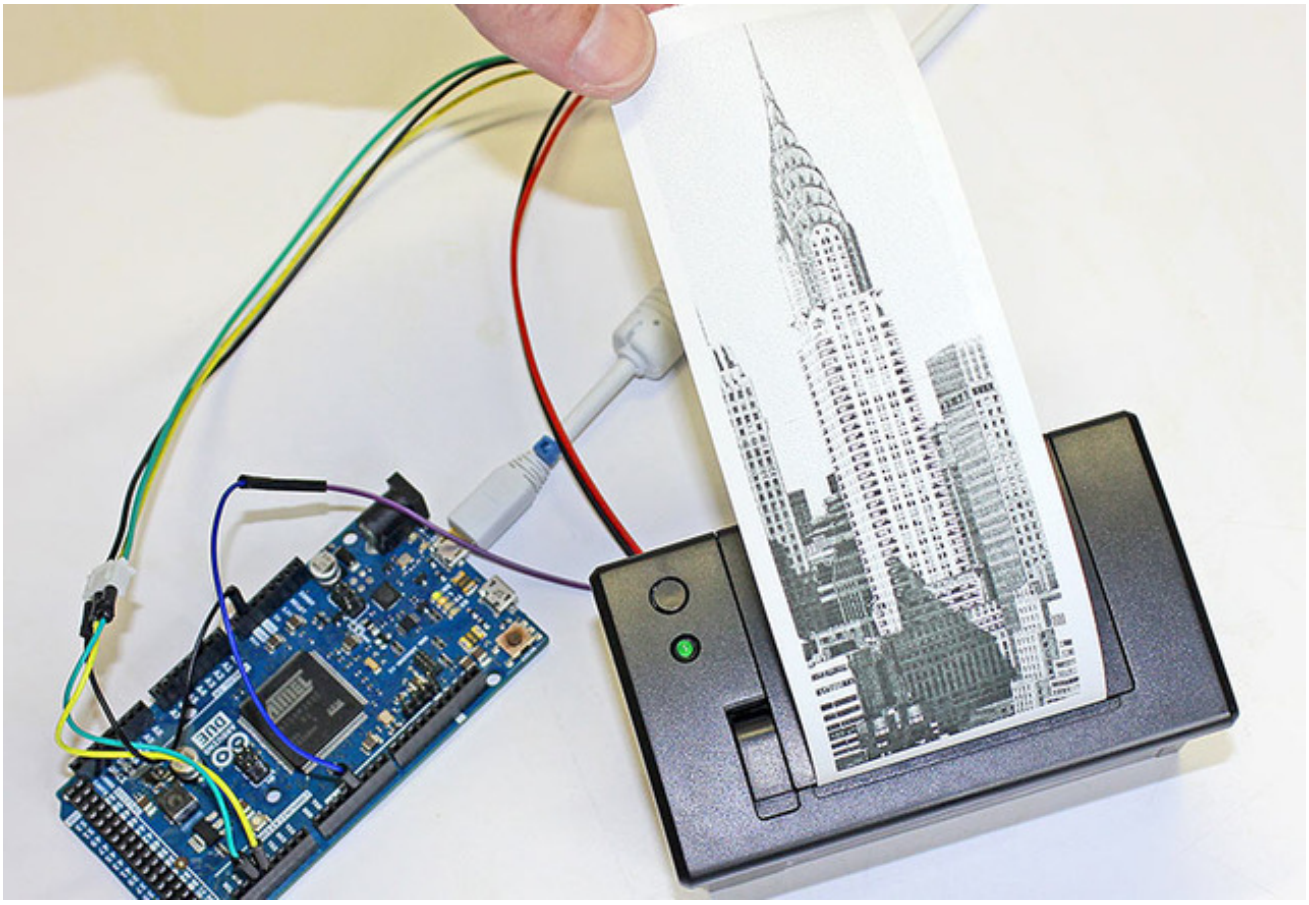
```
Adafruit_Thermal printer(&mySerial, 4);
```

This works just as well with a hardware serial port (e.g. Arduino Mega or Due):

```
Adafruit_Thermal printer(&Serial1, 4);
```

No other changes are necessary. Try this with one of the example sketches…you'll find the printer is suddenly *lots* faster! That's because the software throttle is no longer used…the printer continually reports its actual "ready" state to the microcontroller.

Printing Huge Images



The printBitmap() function can output images from an open stream or stored in PROGMEM (flash memory)…as explained on the “Printing Bitmaps” page.

Although the Arduino Mega has a whopping 256K flash space, a limitation of the AVR microcontroller is that a single array can’t exceed 32K…that’s about a 384x680 pixel bitmap image. If you try to embed a larger image in your code, the compiler will report an error.

One workaround might be to break really long images into multiple smaller images, and print these out consecutively without a feed() in between.

Another is to use a non-AVR Arduino-compatible board, such as the 32-bit Arduino Due. This has no problem with massive arrays. The Chrysler Building image above is 384x1132 pixels!

Other Things to Know

This type of printer fares best with light line art and sometimes dithered photographic images as long as the overall dot density is fairly low, like under 50%. Large solid-filled areas exhibit strange streaky artifacts…this isn’t a bug of the library or printer firmware, but just a side-effect of how receipt printers operate, that they can only heat so many dots at a time and have to pull shenanigans to go beyond that, else they jam.

Here are a couple examples from fancy commercial receipt printers.

Notice in the first one that the “solid black” area isn’t *really* solid black…examining it closely, you can see it’s densely dithered, but not 100% filled.

The second *does* have solid fills, but limits the total area. On any given row, only so many pixels are set.

If you try to print a “dense” image and the paper jams (image gets squashed vertically), pass a lower density value to printer.begin(). Default value is 120. So for example:

```
printer.begin(80);
```

DTR support is not a panacaea. **Glitches occasionally do still happen**…sometimes overflows, sometimes “framing errors” with serial data. But overall it seems fairly reliable and *buttery smooth!*